Graphic LOTOS Specification of an EN50254 System

L. Durante (‡), R. Sisto (‡), and A. Valenzano (†)
{durante, sisto, valenzano}@polito.it
(‡) Dipartimento di Automatica e Informatica, Politecnico di Torino
(†) IRITI – CNR, Politecnico di Torino

***Abstract* - This paper reports on our experience in using a graphical tool for protocol specification based on the G-LOTOS language, the graphical extension of ISO LOTOS, and presents as a case study the development of a formal description of an EN50254 (INTERBUS) factory communication system. The tool is an MS Windows application, thus running on commonly available PC platforms, which lets the user edit a G-LOTOS specification performing on-line checks of syntactic and static semantics rules, generate a corresponding textual LOTOS specification, and invoke an integrated syntax and static semantics analyser. Experience with the tool has shown its usefulness in making industrial protocol specification, editing and understanding an easier task.**

I. INTRODUCTION

Formal methods for the specification of complex software and hardware systems have received considerable attention from the scientific community in the last decade, especially in the areas of safety critical applications such as factory communication systems [1][2][3]. As a consequence, the basic idea that the design and the development activities of concurrent systems, in general, and communication protocols, in particular, can derive substantial benefits from the adoption of rigorous and formal techniques is now widely accepted all over the world.

On the other hand, however, the main appealing features of a formal approach to the specification and design of a system such as the ability to correct specification errors at an early stage, the possibility of checking the design correctness, for instance by simulating the system's behaviour, together with the enhancement of productivity in the design and test phases, heavily depend on the availability of powerful tools that can be viewed as the building blocks of an integrated “computer-aided-specification-development-environment” (CASDE).

In the communication protocols scenario several well-established formal description techniques (FDTs) exist today, which were conceived to reduce the specification effort by exploiting one or more design aspects. In particular, LOTOS [5] consists of an algebraic language that allows the user to describe the system's temporal evolution by means of concurrent processes, behaviour expressions and abstract data types. The advantages of LOTOS are significant because of the expressiveness of the language and its mathematical foundation, but non-experts find it difficult to adopt and, to a certain extent, it is unfamiliar and cannot be considered to be user-friendly. A graphic formalism (graphic syntax) for LOTOS called G-LOTOS [6] was defined at a later date so as to make the language more user-attractive, in order to overcome the kind of problems mentioned above and to enable the user to focus on the structure of the specification rather than being confused by syntactical details.

This paper discusses the use of a graphic tool called GL-Designer for the development of G-LOTOS specifications of factory communication systems, presenting as a case study the specification of a small INTERBUS network. GL-Designer has been conceived and developed at the Computer Engineering Department of the Politecnico of Turin within the framework of ongoing activities oriented to studying and experimenting suitable software tools to build up a CASDE for LOTOS.

II. MODELING THE COMMUNICATION CYCLE

The communication cycle is a critical aspect of the INTERBUS protocol [4], thus it has been specified in LOTOS by means of our graphic tool and then formally verified.

The communication cycle involves three different entities: the master, the slaves and the medium connecting the master to the slaves.

The master acts as a message sender and receiver at the same time while each slave behaves as a repeater that sometimes returns information to the master. Finally, the medium can be thought of as a set of point to point channels that transmit messages and introduce delays. In fact a message transmission with delay can be considered as a read operation followed by a write action.

As the master transmits frames or fill sequences continuously, each section of the ring is always carrying signals i.e. each link connecting two adjacent stations is always active. The same amount of data sent on the medium by a station, must be received from the same link by the next station in the ring. Obviously, each station can receive information from its input connection only if it is able to repeat it on its outgoing link.

A low-level synchronisation of the system could be based on the property explained above i.e. the sequence of receiving and sending operations defines the kind of system timing. A time-tick is defined as a shift of the information stream along the ring.

A.. Synchronisation and timings

LOTOS does not provide any construction to handle time explicitly but, due to the particular features of periodicity and time invariance of the protocol to be modeled, it is possible to overcome such a drawback with the same technique presented in [3]. In fact an INTERBUS ring can be seen as a shift register where all connected stations output a certain amount of bits and, at the same time, they input the same number of bits, thus the sequence of receive and send operations defines the timing of the system and the time-tick can be put into correspondence with the shifting of the information stream along the ring.

The time granularity depends on how frequently atomic steps are performed to shift the information stream along the ring: an accurate choice is, for instance, one step every bit time. In our case study, one word can be assumed as the atomic information unit. In this way, the tasks of reconstructing a word from its bits during the input operations and splitting a word into bits during the output operations is transparent to the G-LOTOS specification.

In this paper we adopt an atomic information unit equal to one word, thus the atomic time-tick is equated to the time needed to transmit a word on the medium, and this choice simplifies the description.

The consequences of our assumption can be negligible for some configurations and/or applications but could be unacceptable for others. However, to remove such a constraint, it is sufficient to select the bit transmission time as the atomic delay and to specify the packing and unpacking operations for translating words into bits and vice versa. In this case, the model and the formalisation of the protocol remain the same: only the time unit has to be changed. The approach followed in this paper simplifies the description without affecting the underlying model.

The timing rule introduced above has been specified in LOTOS by using a synchronisation event that involves all the entities of the specification. The event occurs when each entity performs a read-write or a write-read operation i.e. in each entity of the network the information stream has been shifted by one atomic step.

At the beginning all the stations (master and slaves) execute an output operation, then, after global rendez-vous synchronisation, each station performs an input operation. A second rendez-vous synchronises the stations once again so that a network step is completed.

III. GRAPHICAL SPECIFICATION OF AN INTERBUS SYSTEM

Figure 1 shows the system configuration described by our LOTOS specification: there is a ring with a master (M), two slaves (S1 and S2), and three channels (C) interconnecting the nodes. The inps and outs events take into account the different stations behaviours i.e. outM is an output performed by the master, inp1 is an input performed by the salve 1 and so on. Thus each channel performs an input by synchronising on an out event. The circular, dotted arrow highlights the data flow in the ring.

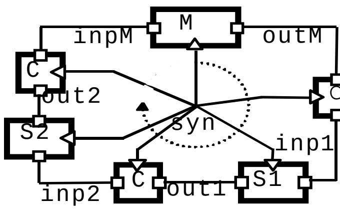
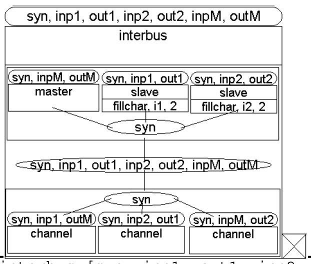


Fig. 1 - Topology model..

The syn gate is needed for timing synchronisation: all stations perform an output operation and a global rendez-vous occurs at the gate syn, then all stations perform an input and another rendez-vous occurs at syn. In this way two input operations which are not interleaved by an output (or vice versa) cannot occur and the data stream is transferred along the ring step by step. A syn rendez-vous can be interpreted as a clock-tick. A complete step of the data stream is made up of an output and input pair with two syn rendez-vous taking place.

Our specification describes the behaviour of the system after the initialisation phase (that is in a steady state condition), when each slave already knows the position of its data inside the master's frame. In the following the graphical specification is introduced together with the corresponding textual form.



```
specification interbus [syn, inp1, out1, inp2, out2, inpM,
outM]: noexit
  library Boolean, NaturalNumber, Word endlib
behaviour
  (master[syn,inpM,outM)
  |[syn]|
  slave[syn,inp1,out1](fillchar,i1,2)
  |[syn]|
  slave[syn,inp2,out2](fillchar,i2,2))
  |[syn, inp1, out1, inp2, out2, inpM, outM]|
  (channel[syn,inp1,outM]
  |[syn}|
  channel[syn,inp1,outM]
  |[syn}|
  channel[syn,inp1,outM])
where
  …
endspec
```

Fig. 2 - The specification structure.

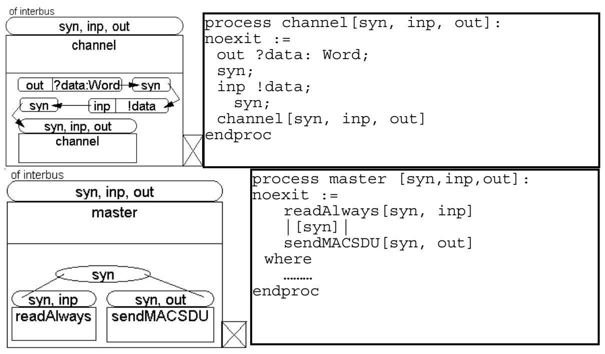
A.. Specification structure

Figure 2 shows the graphical and the textual description of the specification structure based on a full synchronisation between the set of nodes and channels.

Processes inside each box (the master, the two slaves, and the three channels) are synchronised only on the gate syn, in fact there is no direct connection between the boxes describing the stations (between any pair of nodes there is a channel and vice versa). The gate names are the same as in Figure 1, while abstract types Boolean, NaturalNumber and Word have been defined in an external library file.

The focus of our specification is on the data communication aspects and not on the data processing. Thus all the different kinds of words have been defined as LOTOS nullary operators of a single abstract data type.

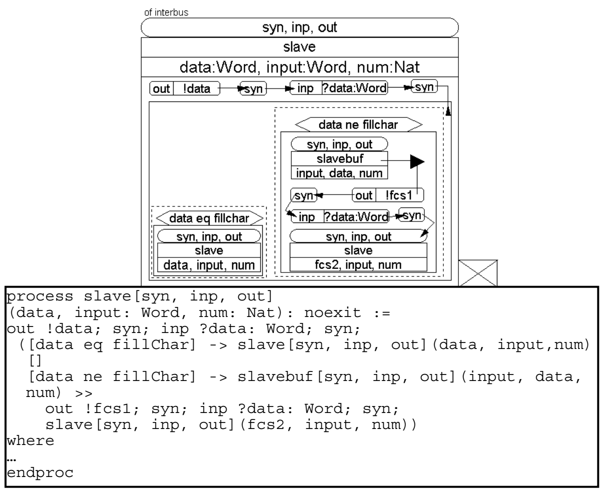
The two slaves are parameterised by means of fillchar, i1, 2 and fillchar, i2, 2, where fillchar is the fill sequence that each slave outputs at the beginning of the operations, i1 and i2 correspond respectively to inp (1) and inp (2) in Figure 1, (i.e. data that the slaves send to the master); 2 is the number of words in the master frame that precede out (1) and out (2) respectively. Such a number is computed starting with the first word of the frame received by a slave.



```
process channel[syn, inp, out]:
noexit :=
 out ?data: Word;
 syn;
 inp !data;
   syn;
 channel[syn, inp, out]
endproc
```

```
process master [syn,inp,out]:
noexit :=
    readAlways[syn, inp]
    |[syn]|
    sendMACSDU[syn, out]
  where
    .........
endproc
```

Fig. 3 - Channel and master processes.



```
process slave[syn, inp, out]
(data, input: Word, num: Nat): noexit :=
out !data; syn; inp ?data: Word; syn;
 ([data eq fillChar] -> slave[syn, inp, out](data, input,num)
  []
  [data ne fillChar] -> slavebuf[syn, inp, out](input, data,
  num) >>
    out !fcs1; syn; inp ?data: Word; syn;
    slave[syn, inp, out](fcs2, input, num))
where
...
endproc
```

Fig. 4 - The slave process.

B. The channel process

Figure 3 shows the structure of the process describing the channel. The first action performed by the system is a station output on gate out, which corresponds to a data input on the same gate: the data variable is read and, after the syn event, the same data is passed to the next station. After another syn event occurs, the cycle is restarted: this procedure is handled by the recursive instantiation of the process.

C. The master process

The activity of the master of Figure 3 has been di vided into two independent tasks: readAlways which reads data coming from the ring, and sendMACSDU which sends the summation-frame on every cycle. The two processes are fully independent, since we are interested in the communication mechanism only, and in the processing activity local to each station.

D. The slave process

The slave is the most important process of the whole specification: it behaves as a repeater (recursively recalling) when it receives fill sequence. The end of the fill pattern is used to detect the beginning of the summation-frame, so process slavebuf is entered in order to send data to the master and to read data coming from the master. When such an operation is completed, a new FCS word (fcs1) is transmitted, followed by fcs2, then the slave process is restarted.

IV. EXPERIMENTAL RESULTS

The specification has been translated into a textual form and has been analysed by a logical simulation tool [9] which proved that the frame sent by the master, the one received and then sent by each slave, and the one received by the master are exactly those foreseen in [4].

The tool described in this paper was tested not only with the INTERBUS case study, but also taking some other well-known G-LOTOS specifications as test cases.

The tests were aimed both at measuring some of the tool's numerical performance figures and at obtaining a preliminary assessment of some of its qualitative and functional features such as user-friendliness and to what extent the tool facilitates the designer in developing a specification. All the tests were conducted by assigning the job of developing the graphical specifications to undergraduate students with some minimal knowledge of LOTOS. They succeeded in their job after a short training period. For what concerns numerical results, we measured some parameters related to speed and memory requirements. All tests were done on a low cost PC with 48MB of main memory using a 90MHz Intel Pentium processor and the Windows NT Server 4.0 operating system. The results are collected in Table 1.

For each test case the table reports the total number of graphical blocks making up the specification and the number of lines in the textual LOTOS version. For what concerns memory requirements, we measured and reported the binary file dimensions (Grf File) and the amount of main memory used. The latter was measured by means of the Windows NT Task Manager that gives the amount of memory allocated by each running process.

More precisely, the values obtained are computed as the difference between the amount of memory occupied by the tool process before and after the file was loaded. For what concerns speed, we measured the time needed to load the graphical file, and the time needed to generate the corresponding textual LOTOS specification.

Test case	# Blocks	# lines	Grf File size (Kbytes)	Memory (Kbytes)	Time to load (seconds)	Time to convert (seconds)
INTERBUS	10	200	28	1704	3	1
BRP Service	9	150	14	1540	2	0
CCR Service	39	658	138	2068	5	2
TP Service	87	1129	360	3184	9	4

Tab. 1 - Numerical Experimental results.

IV. EXPERIMENTAL RESULTS

The GL-designer project has been aimed at developing and experimenting a graphical environment designed to create LOTOS specifications and to demonstrate its feasibility using conventional low-cost hardware and software resources such as Personal Computers and MS Windows operating systems. The prototype currently running on some machines in our department has been tested by designing formal G-LOTOS descriptions for some typical and well-known medium and large size standard protocols, including factory communication protocols. We have also verified that non-expert users such as undergraduate students in computer engineering are able, after a short period of training, to develop quite complex specifications by focusing primarily on their hierarchical organisation rather than being bored with a number of syntactical and unessential details. Numerical experimental results show that the tool can deal with large specifications with reasonable performance.

The GL-designer can be considered as a building block for a CASDE system, even though further activities have still to be done to improve some functionalities and to enhance its flexibility. In particular we plan to extend the capabilities of GL-designer so as to automatically generate graphic representations of (pre-existing) textual LOTOS specifications and to integrate other LOTOS tools such as simulators in it. Furthermore additional investigations will be carried out in order to port the editor on different machines and operating systems, by using other types of programming supports such as Java.

To our knowledge, only two other graphical tools supporting in some way the G-LOTOS formalism have ever been developed: the ELUDO toolkit [7], developed at the University of Ottawa, and GLD (Graphical LOTOS Designer), developed at UPM, Madrid [8]. Both tools run on Unix platforms and are based on the X-Window system.

REFERENCES

[1] L. Durante, R. Sisto, A. Valenzano, “Formal Specification and Verification of the Real-time Scheduler in FIP”, in Proc. WFCS95, 1st IEEE Workshop on Factory communications Systems, Leysin, Switzerland, October 1995, pp. 99-106.

[2] N. Petalidis, D S. Gill, "The formal Specification of the Fieldbus Foundation Link Scheduler in E-LOTOS", in Proc. 2nd International Conference on Formal Engineering Methods, 1998.

[3] L. Durante, R. Sisto, and A. Valenzano, "A LOTOS Specification of the SERCOS Field-bus Protocol", in Proc. 6th International Conference on Software Engineering and Knowledge Engineering, 1994.

[4] CENELEC, "High Efficiency Communication Subsystem for Small Data Packages", Final Draft EN 50254, 1997.

[5] ISO “Information Processing Systems - Open Systems Interconnection - LOTOS - a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour”, IS N. 8807, 1989.

[6] ISO/IEC “ Information Processing Systems - Open Systems Interconnection – LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour – Amd. 1: G-LOTOS”, JTC 1/SC 21/N. 8913, Oct. 1994.

[7] The Xeludo User Manual, http://LOTOS.csi.UOttawa.ca/eludo/usrman.

[8] GLD V.1.2b Graphical Designer for LOTOS User Manual, Dept. Of Telematic Systems Engineering, Technical University of Madrid, Spain, 1994.

[9] DAI/Polito LOTOS-C Compiler, http://www.dai-arc.polito.it/dai-arc/auto/tools/tool3.shtml.